Creating a TCK Module

Creating a JSR 362 test case module and making the tests visible to the test driver

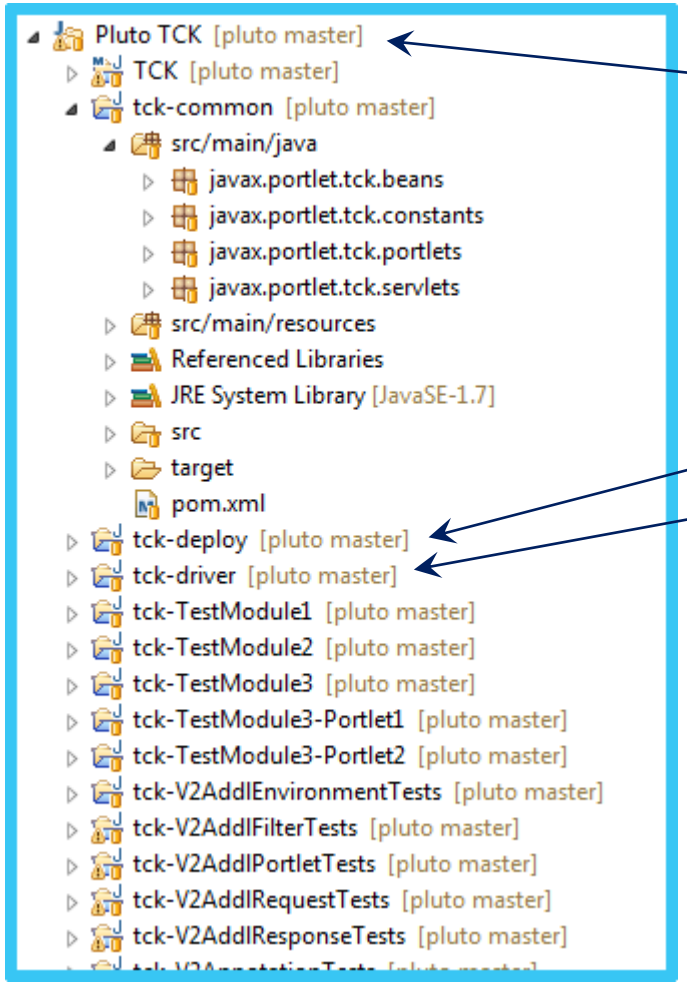
TCK Goal

- Wide test coverage to assure that most or all JSR 362 attributes are implemented
- Test cases should be simple; described in 1 sentence:
  - "The MutableRenderParameters.setValues method throws an IllegalArgumentException if the parameter name is null."
  - "The MutableRenderParameters.setValues method accepts an empty array as values array argument."
- Should test only JSR 362 functionality – no Pluto specifics.
- Concentrate on mandatory functionality as described by the spec.

How the TCK Works

- The test cases are implemented in portlets and sometimes servlets or JSPs.
- These are deployed on the target portal / portlet container
- The test driver:
  - sends http requests to the portal under test
  - analyzes the response
  - extracts test results produced by the TCK portlets
    - Test results designated by special HTML tag 'name' and 'id' attribute values
  - based on Junit / Selenium
- The TCK:
  - Consists of independent modules
  - Each module defines list of test cases, portlets, and page definitions
  - Provides common tools for creating markup that the driver can understand
- The TCK Build process:
  - Aggregates test cases lists & page definitions over all TCK modules
  - Collects deployable artifacts in central location
  - For Pluto, use the pluto profile: build with `'mvn clean install –Ppluto'`

Project Structure



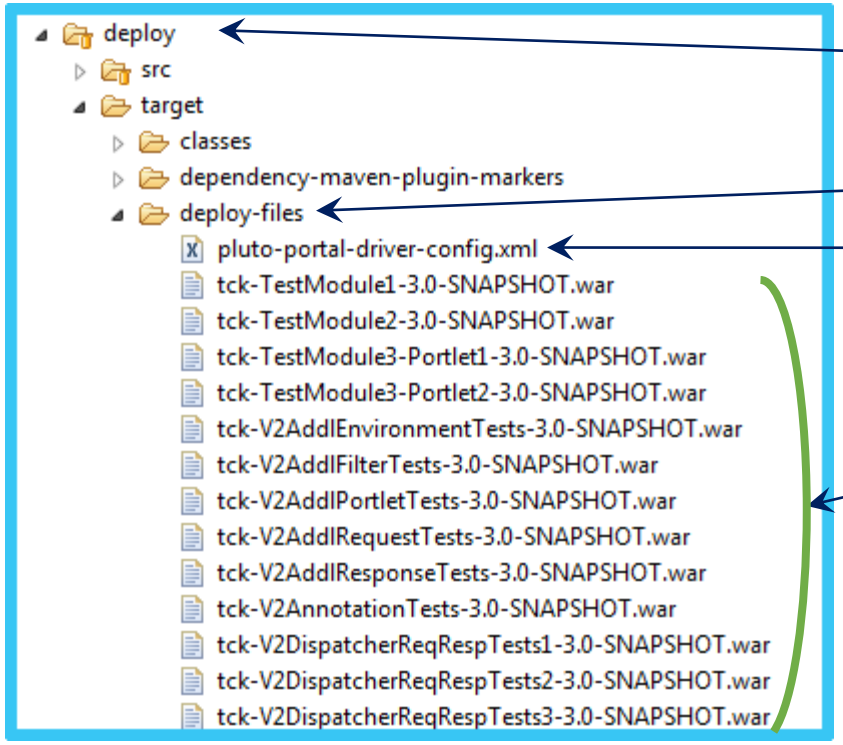
Common utilities used by multiple test modules

Contains deployment artifacts

Contains the test driver

The test modules

Deploy Directory Structure (after build)



Deploy directory

Contains deployment artifacts

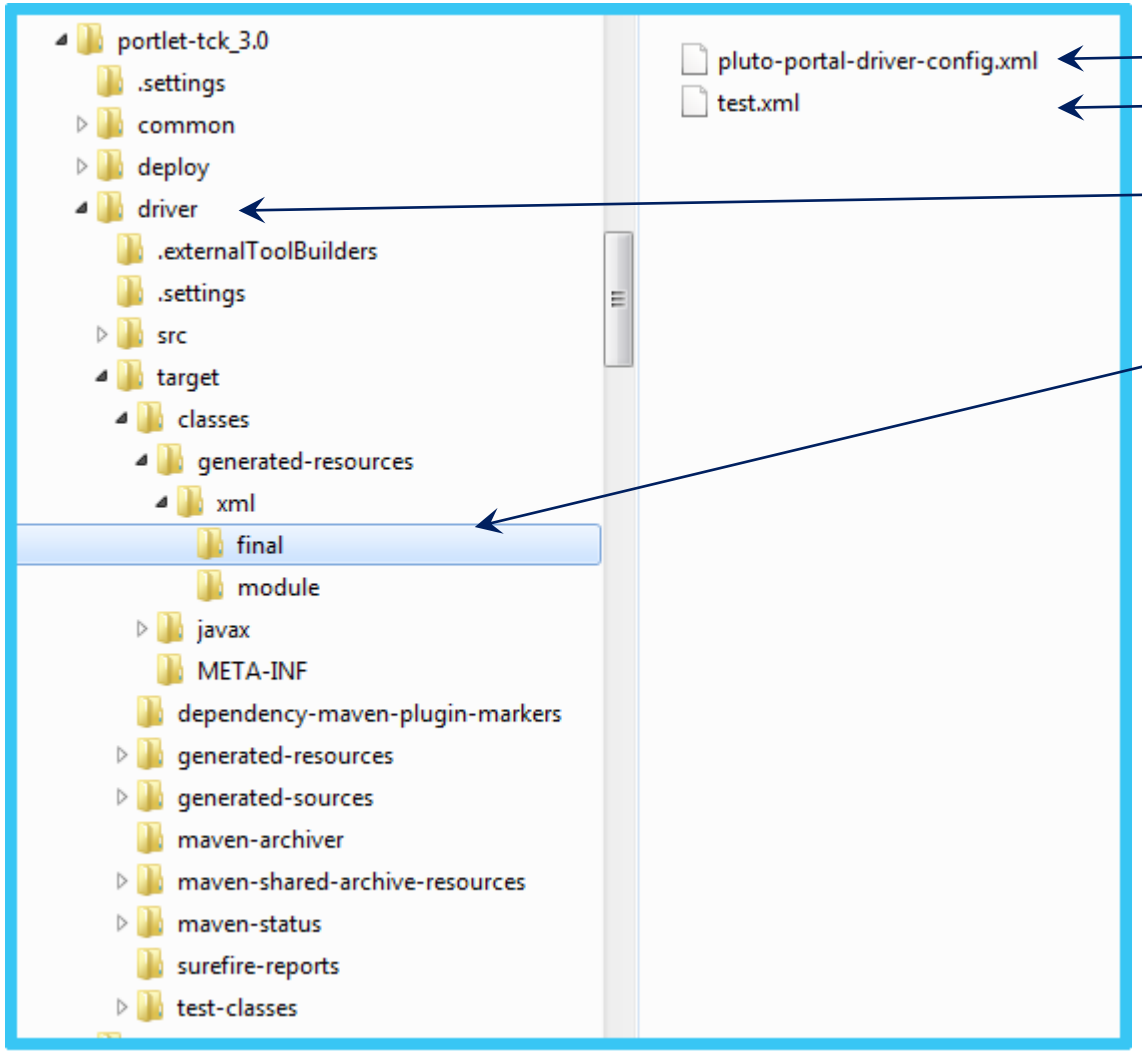
The page file

Portlet war files

The page file is an XML file containing the page definitions expected by the TCK driver. Each page definition defines a page name along with the portlets expected to be on the page. The format can be used directly with Pluto.

It is expected that portal vendors will use some sort of XML transformation to create corresponding pages on their systems.

Driver Directory Structure (after build)



The page file

The test case list file

Driver directory

This is where you find the test case list

The Test Case List

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<entry key="TestModule1_PortletCanBeRendered">TestModule1_PortletCanBeRendered</entry>
<entry key="TestModule1_PortletConfigTest">TestModule1_PortletConfigTest</entry>
<entry key="TestModule1_GetInitParameterNamesTest1">TestModule1_GetInitParameterNamesTest1</entry>
<entry key="TestModule1_GetInitParameterNamesTest">TestModule1_PortletConfigTest</entry>
<entry key="TestModule1_GetSupportedLocalesTest">TestModule1_PortletConfigTest</entry>
<entry key="TestModule1_GetDefaultNamespaceTest">TestModule1_PortletConfigTest</entry>
<entry key="TestModule1_GetPortletNameTest">TestModule1_GetInitParameterNamesTest1</entry>
<entry key="TestModule1_GetSupportedLocalesTest1">TestModule1_GetInitParameterNamesTest1</entry>
<entry key="TestModule2_CanCreateRenderURL">TestModule2_SomeLinkTests</entry>
<entry key="TestModule2_CanGetPortletSession">TestModule2_SomeLinkTests</entry>
<entry key="TestModule2_CanSetAttributeInPortletSession">TestModule2_SomeLinkTests</entry>
<entry key="TestModule2_RenderURLWithParameterCanBeClicked">TestModule2_SomeLinkTests</entry>
<entry key="TestModule2_PublicRenderParameterCanBeSetThroughUrl">TestModule2_SomeLinkTests</entry>
<entry key="TestModule3_PublicRenderParameterTestDifferentPortletApplications">TestModule3_2PortletAppsTest</entry>
<entry key="TestModule3_PublicRenderParameterTestDifferentQName">TestModule3_2PortletAppsTest</entry>
<entry key="TestModule3_PublicRenderParameterTestDifferentIdentifier">TestModule3_2PortletAppsTest</entry>
<entry key="V2AnnotationTests_ProcessAction_ApiAction_name">V2AnnotationTests</entry>
<entry key="V2AnnotationTests_ProcessEvent_ApiEvent_qname">V2AnnotationTests</entry>
<entry key="V2AnnotationTests_ProcessEvent_ApiEvent_name">V2AnnotationTests</entry>
<entry key="V2AnnotationTests_RenderMode_ApiRender_name">V2AnnotationTests</entry>
<entry key="V2EnvironmentTests_CacheControl_ApiRender_getExpirationTime1">V2EnvironmentTests</entry>
<entry key="V2EnvironmentTests_CacheControl_ApiRender_getExpirationTime2">V2EnvironmentTests</entry>
<entry key="V2EnvironmentTests_CacheControl_ApiRender_getExpirationTime3">V2EnvironmentTests</entry>
```

Test case names

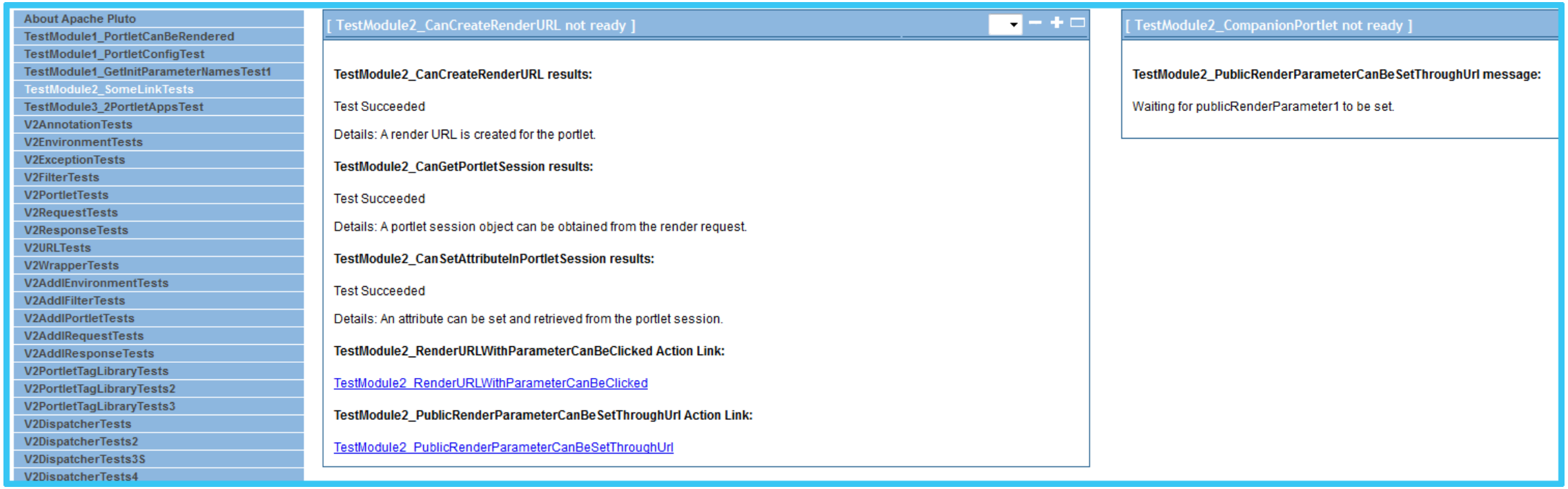
Page names

The test case list is an XML file in 'properties file' format. The property key is the test case name, and the property value is the page name on which the test case must be found. The test case page names must match page names defined in the page file. The test driver reads in the test case list to get the tests to be executed.

When the test driver wants to execute a test, it looks for a link on the page that contains the page name. It clicks that link to access the page on which the test case is to be found.

How it Looks on a Real Portal Page

```
12 <entry key="TestModule2_CanCreateRenderURL">TestModule2_SomeLinkTests</entry>
13 <entry key="TestModule2_CanGetPortletSession">TestModule2_SomeLinkTests</entry>
14 <entry key="TestModule2_CanSetAttributeInPortletSession">TestModule2_SomeLinkTests</entry>
15 <entry key="TestModule2_RenderURLWithParameterCanBeClicked">TestModule2_SomeLinkTests</entry>
16 <entry key="TestModule2_PublicRenderParameterCanBeSetThroughUrl">TestModule2_SomeLinkTests</entry>
```



The Driver

- Executed through maven: `'mvn test -Prun-tck'`
- Configured in the TCK master POM file
  - Options: login URL, user name, password, fields for user name & password, Selenium browser type, timeout

```
<!-- Configuration of URL to login page -->
<test.server.login.url>http://localhost:8080/pluto/portal/About Apache Pluto</test.server.login.url>

<!-- Configuration of portlet container under test for generated URLs -->
<!-- (only needed if test.url.strategy=generateURLs) -->
<test.server.host>localhost</test.server.host>
<test.server.port>8080</test.server.port>

<!-- HTML field IDs and values for username & password to enable automatic  login -->
<test.server.username.id>j_username</test.server.username.id>
<test.server.username>pluto</test.server.username>
<test.server.password.id>j_password</test.server.password.id>
<test.server.password>pluto</test.server.password>

<!-- Specifies the browser to be used by selenium WebDriver for running the tests. -->
<!-- Can be used with firefox or HTMLUnit without setting the test.browser.webDriver property. -->
<!-- Use of Chrome or IE requires the webDriver to be downloaded and available. Set the -->
<!-- test.browser.webDriver property to point to the appropriate WebDriver server. -->
<test.browser>firefox</test.browser>
<test.browser>HTMLUnit</test.browser>
<!-- commented outline below shows configuration for the Internet Explorer driver -->
<!-- test.browser.webDriver>C:\ntutil\IEDriverServer_x64_2.42.0\IEDriverServer.exe</test.browser.webDriver -->

<!-- Specify timeout in seconds for the driver to wait for page load. must be an integer. -->
<test.timeout>3</test.timeout>
```

In general, ignore

Recommendation: stay with HtmlUnit, it's fastest and seems to work pretty well.

Some Driver Command Line Options

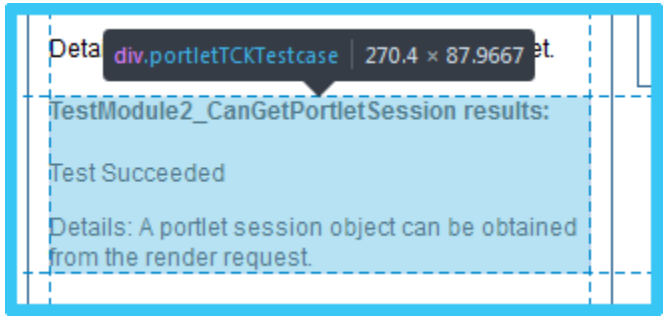
- `'mvn test -Prun-tck –Dtest.timeout=<integer>'`
  - Sets the timeout to the specified integer value
- `'mvn test -Prun-tck –Dtest.module=<string>'`
  - The driver will only execute those test cases whose test case name contains the specified string.
  - By making the string more specific, you can execute a group of test cases or even just a single test case.
  - `'mvn test -Prun-tck –Dtest.module=V3PortletContextTests'`
    - Executes all test cases in module 'V3PortletContextTests'
  - `'mvn test -Prun-tck –Dtest.module=V3PortletContextTests_Context_getClassLoader'`
    - Executes the single test case '`V3PortletContextTests_Context_getClassLoader`'
- `'mvn test -Prun-tck –Dtest.debug=true'`
  - Causes the driver to write (very considerable amounts of) debug information to stdout.

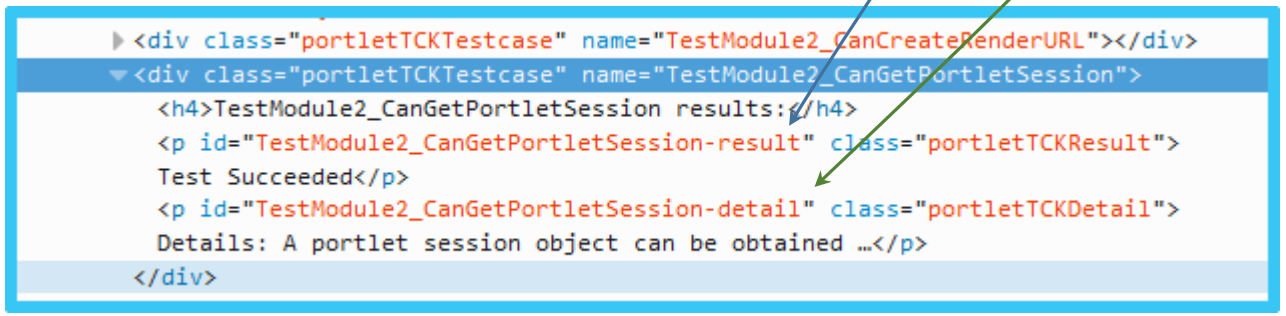
Driver Execution

- The driver initializes
  - Reads test case file
  - Sort test cases according to page (performance optimization)
  - Filters test cases by name if required
- Conceptually, the driver executes each test case as follows:
  1. Determines the page on which the test case is to be found
  2. Looks for the link containing the page name on the portal page & clicks it
     - If the link can't be found, accesses the login page, logs in, tries again
     - Searches by link text, not by tag attribute
  3. Looks for a test 'setup' link and
     - If found, clicks it and waits for page refresh
     - A 'setup' link can be an anchor link (GET) or a form submission button (POST)
  4. Looks for a test 'execute' link and clicks it if found; waits for refresh
     - If found, clicks it and waits for page refresh
     - An 'execute' link can be an anchor link (GET) or a form submission button (POST)
  5. Looks for test results
     - Extracts results from markup
     - Determines success / failure
  6. Goes to next test case

How the Driver finds Test Case Markup

- It searches page markup for HTML tags by 'id' attribute
  - The id is based on the test case name
  - Test results: id='testCaseName-result'
  - Test details: id='testCaseName-detail'
  - Setup link / button: id='testCaseName-setup'
  - Execute link / button: id='testCaseName-clickme'
  - Resource link: id='testCaseName-reslink' (not directly used by driver)
  - Resource div: id='testCaseName-resdiv' (not directly used by driver)
- Example:





The Test Case

- A test case consists of test case results and test case details
- It is identified by a test case name (aka ID)
- The name should be a valid Java identifier; most importantly cannot contain blanks
  - Must be unique within the entire TCK (!) ... Please follow naming convention
- The details contains the test case description
  - And possibly information collected at run time
- Convention for test case name:
  - Allows the class that contains a failing test case to be easily found
  - (please follow!)

Module (file system name)

Specific test case suffix
(unique within class)

version

V2EnvironmentTests_CacheControl_ApiRender_getExpirationTime2

Class name (unique within module)

Utilities for generating Test Case markup

- The TCK common module provides utilities for generating markup
  - The TCK module developer is STRONGLY ENCOURAGED to use them!
    - Allows developers to orient themselves easily in test case modules
    - Allows changes to be made in single location if it should become necessary

The TestCaseDetails Class

```
javax.portlet.tck.beans
TestCaseDetails
    detailsMap : Map<String, String>
    TestCaseDetails()
    TestCaseDetails(Map<String, String>)
    clear() : void
    containsKey(Object) : boolean
    containsValue(Object) : boolean
    entrySet() : Set<Entry<String, String>>
    get(Object) : String
    isEmpty() : boolean
    keySet() : Set<String>
    put(String, String) : String
    putAll(Map<? extends String, ? extends String>) : void
    remove(Object) : String
    size() : int
    values() : Collection<String>
    getTestResultFailed(String) : TestResult
    getTestResultSucceeded(String) : TestResult
```

- The `TestCaseDetails` class maps test case names to test case descriptions.
- Has methods for getting a test result based on the test case name.
- Extend `TestCaseDetails` to provide a map for your test case names / details.
- If used only by one module, keep it in that module.
- If common to several modules, put it in the common module.

The TestResult Class

```
javax.portlet.tck.beans
TestResult
    tcName : String
    tcDetail : String
    tcSuccess : boolean
    TestResult()
    TestResult(String, boolean, String)
    getTcName() : String
    setTcName(String) : void
    getTcDetail() : String
    setTcDetail(String) : void
    isTcSuccess() : boolean
    setTcSuccess(boolean) : void
    appendTcDetail(String) : void
    appendTcDetail(Throwable) : void
    toString() : String
    writeTo(Writer) : void
```

- The `TestResult` class encapsulates a test case name / test case description pair.
- Generates markup for success / failure completion - see `toString()` and `writeTo()`.
- The setTCSuccess() method allows test case success / failure to be recorded.
- The appendTCDetail(String) method allows details (such as failure reason) to results at runtime.
- The appendTCDetail(Throwable) method adds stack trace info from a Throwable to the result if the portlet API method tested throws an unexpected exception.

```java
/* TestCase: V3PortletContextTests_Context_getContextPath */
/* Details: "The PortletContext.getContextPath method returns the */
/* context path for this portlet application." */
{
   TestResult result = tcd.getTestResultSucceeded(V3PORTLETCONTEXTTESTS_CONTEXT_GETCONTEXTPATH);
   String path = portletContext.getContextPath();
   result.appendTcDetail("Context path: " + path);
   if (path == null) {
      result.setTcSuccess(false);
      result.appendTcDetail("Context path was null.");
   }
   result.writeTo(writer);
}
```

The TestLink and TestSetupLink Classes

```
TestLink
    TestLink()
    TestLink(String, PortletURL)
    TestLink(String, String)
    toString() : String
    writeTo(Writer) : void
```

```
TestSetupLink
    TestSetupLink()
    TestSetupLink(String, PortletURL)
```

- The `TestLink` class encapsulates a test execution link.
- Generates `<a>` tag markup - see `toString()` and `writeTo()`.
- The `TestLink` class encapsulates a test setup link.
- Generates `<a>` tag markup - see `toString()` and `writeTo()`.
- Both are generally used with render URLs, and should not be used with action URLs.
- Parameters can be set on the URL before passing it to the link class.

```
} else {
   PortletURL aurl = portletResp.createRenderURL();
   aurl.setParameter(BUTTON_PARAM_NAME, V2ADDLRESPONSETESTS_SPEC2_12_ACTION_COOKIE3);
   TestSetupLink tb = new TestSetupLink(V2ADDLRESPONSETESTS_SPEC2_12_ACTION_COOKIE3, aurl);
   tb.writeTo(writer);
}
```

The TestButton and TestSetupButton Classes

```
TestButton
    TestButton()
    TestButton(String, PortletURL)
    TestButton(String, String)
    toString() : String
    writeTo(Writer) : void
```

```
TestSetupButton
    TestSetupButton()
    TestSetupButton(String, PortletURL)
```

- The `TestButton` class encapsulates a test execution form which is submitted when the button is clicked.
- Generates `<FORM>` tag markup - see `toString()` and `writeTo()`.
- Sets an action parameter with name `Constants.BUTTON_PARAM_NAME` to the value 'testCaseName + Constants.CLICK_ID' for controlling test execution
- The `TestSetupButton` class encapsulates a test setup form which is submitted when the button is clicked.
- Generates `<FORM>` tag markup - see `toString()` and `writeTo()`.
- Sets an action parameter with name Constants.BUTTON_PARAM_NAME to the value 'testCaseName + Constants.CLICK_ID' for controlling test execution
- Both must be used with action URLs.
- Parameters can be set on the URL before passing it to the button class.

```
PortletURL aurl = portletResp.createActionURL();
TestButton tb = new TestButton(V2ADDLRESPONSETESTS_SPEC2_12_ACTION_COOKIE3, aurl);
tb.writeTo(writer);
```

The ResourceLink class

```
ResourceLink
  ResourceLink()
  ResourceLink(String, ResourceURL)
  ResourceLink(String, String)
  toString() : String
  writeTo(Writer) : void
  writeResourceFetcher(Writer) : void
  writeResourceFetcherImmediate(Writer) : void
```

- The ResourceLink class encapsulates a resource URL
- Generates JavaScript to fetch the resource
- The writeResourceFetcher method writes a link and JavaScript code that, when the link is clicked clicked, fetches the resource and inserts it into the DOM
- The writeResourceFetcherImmediate method writes a link and JavaScript code that immediately fetches the resource and inserts it into the DOM

```java
/* TestCase: V2URLTests_BaseURL_ApiRenderResurl_setParameterA5 */
/* Details: "Method setParameter(String, String): A resource */
/* parameter can be set" */
TestResult tr1 = tcd.getTestResultFailed(V2URLTESTS_BASEURL_APIRENDERRESURL_SETPARAMETERA5);
try {
   ResourceURL turl = portletResp.createResourceURL();
   turl.setParameter("tc", V2URLTESTS_BASEURL_APIRENDERRESURL_SETPARAMETERA5);
   turl.setParameter("parm1", "val1");

   // add the resource results fetcher to the output stream
   ResourceLink rl = new ResourceLink(V2URLTESTS_BASEURL_APIRENDERRESURL_SETPARAMETERA5, turl);
   rl.writeResourceFetcher(writer);
} catch (Exception e) {
   tr1.appendTcDetail(e);
   tr1.writeTo(writer);
}
```

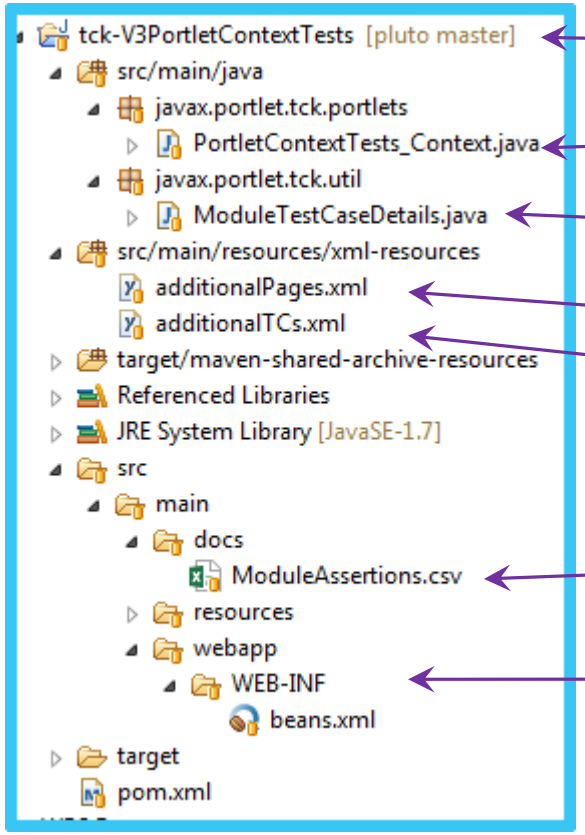
The CompareUtils & Constants classes

```
CompareUtils
  stringsEqual(String, String, TestResult) : void
  stringsEqual(String, String, String, String, TestResult) : void
  arraysEqual(String, String[], String, String[], TestResult) : void
  enumsEqual(String, Enumeration<? extends Object>, String, Enumeration<? extends Object>, TestResult) : void
  mapsEqual(String, Map<String, String[]>, String, Map<String, String[]>, TestResult) : void
  setsEqual(String, Set<? extends Object>, String, Set<? extends Object>, TestResult) : void
```

- Contains some static methods for String, Enumeration, Map, etc. comparison.
- javax.portlet.tck.constants.Constants in the common module defines a bunch of sometimes useful constants.

- If you have more ideas for common tools, feel free to add them!

V3 Test Module Basic Structure



Maven artifact name

Class containing test cases

Class containing test case definitions. If you define a bunch of test cases used in more than one module, put the map file in the common module

Page definitions (don't change name, or the build process won't find it)

Test case list (don't change name, or the build process won't find it)

Test case assertions file (optional). CSV file containing info about the test cases.

Portlet.xml, web.xml not required. Beans.xml required if you use CDI features

The test list and page files

Page definition file:

```
1 <?xml version="1.0" encoding="UTF-8"?><pluto-portal-driver xmlns="http://portals.apache.org/pluto
2 <portal-name>pluto-portal-driver</portal-name>
3 <portal-version>2.1.0-SNAPSHOT</portal-version>
4 <container-name>Pluto Portal Driver</container-name>
5 <supports>
6 <portlet-mode>view</portlet-mode>
7 <portlet-mode>edit</portlet-mode>
8 <portlet-mode>help</portlet-mode>
9 <portlet-mode>config</portlet-mode>
10 <window-state>normal</window-state>
11 <window-state>maximized</window-state>
12 <window-state>minimized</window-state>
13 </supports>
14 <render-config default="About Apache Pluto">
15
16 <page xmlns="" name="V3PortletContextTests" uri="/WEB-INF/themes/pluto-default-theme.jsp">
17
18 <portlet context="/tck-V3PortletContextTests-3.0-SNAPSHOT" name="PortletContextTests_Context"/>
19 </page>
20 </render-config>
21 </pluto-portal-driver>
```

Test case list file:

```
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<!-- JSR 286 API PortletContextTests test case names and page mappings -->
<entry key="V3PortletContextTests_Context_getClassLoader">V3PortletContextTests</entry>
<entry key="V3PortletContextTests_Context_getContextPath">V3PortletContextTests</entry>
<entry key="V3PortletContextTests_Context_getEffectiveMinorVersion">V3PortletContextTests</entry>
<entry key="V3PortletContextTests_Context_getEffectiveMajorVersion">V3PortletContextTests</entry>
</properties>
```

The page name defined in the page definition file must match the page name used in the test case list file.

Recommendation:
Use the module name according to the test case name convention.

Creating a new Test Module

- Create your module directory
  - Please prefix your module name with 'V3' for ease of sorting
  - Example: 'V3PortletContextTests'
- Create your POM
  - For most tests, you should be able to copy the POM from 'V3PortletContextTests' and just change the artifact ID.

```
23    <parent>
24       <groupId>javax.portlet</groupId>
25       <artifactId>portlet-tck</artifactId>
26       <version>3.0-SNAPSHOT</version>
27    </parent>
28
29    <artifactId>tck-V3PortletContextTests</artifactId>
30
31    <packaging>war</packaging>
```

Copy the POM from an existing module and change the artifact ID. Note that the artifact ID within the TCK is not the same as the module name, which in retrospect was probably a poor design decision. Please use the module name prefixed with 'tck-'.

Adding a New Module to the TCK Build

- This is a pain. There are too many moving parts, and it's easy to forget something.
  - If you have ideas / time for improvements, feel free ...
- You need to edit the following files (wrt the TCK root directory):
  - ./pom.xml
  - ./deploy/pom.xml
  - ./driver/pom.xml (2 locations!)
  - ./driver/src/main/resources/xml-resources/pageFiles.xml
  - ./driver/src/main/resources/xml-resources/testFiles.xml

Adding a New Module to the TCK Build

*TCK/pom.xml

```
126    <modules>
127       <module>common</module>
128       <module>TestModule1</module>
129       <module>TestModule2</module>
130       <module>TestModule3</module>
131       <module>TestModule3-portlet1</module>
132
133       <module>V3PortletContextTests</module>
134
135       <module>deploy</module>
136       <module>driver</module>
137    </modules>
```

Add the module name to the modules list in the TCK POM.

tck-deploy/pom.xml

```
287       </dependency>
288
289       <dependency>
290          <groupId>${project.groupId}</groupId>
291          <artifactId>tck-V3PortletContextTests</artifactId>
292          <version>${project.version}</version>
293          <type>war</type>
294       </dependency>
295
296       <dependency>
297          <groupId>${project.groupId}</groupId>
298          <artifactId>tck-driver</artifactId>
299          <version>${project.version}</version>
300          <type>jar</type>
301       </dependency>
302    </dependencies>
```

Add the new module as a dependency to the deploy subproject POM. Be sure to use the corresponding **artifact ID** which can be different than the module ID.

Adding a New Module to the TCK Build

*tck-driver/pom.xml

```
313            <type>war</type>
314         </dependency>
315
316         <dependency>
317            <groupId>${project.groupId}</groupId>
318            <artifactId>tck-V3PortletContextTests</artifactId>
319            <version>${project.version}</version>
320            <type>war</type>
321         </dependency>
```

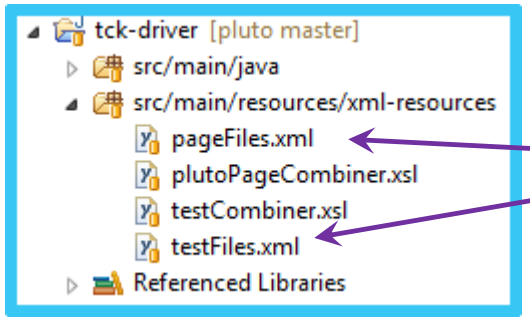
Add the new module as a dependency to the driver subproject POM. Be sure to use the corresponding **artifact ID** which can be different than the module ID.

*tck-driver/pom.xml

```
345            <plugin>
346               <groupId>org.apache.maven.plugins</groupId>
347               <artifactId>maven-dependency-plugin</artifactId>
348               <executions>
349                  <execution>
350                     <id>xml-resource-dependencies</id>
351                     <phase>generate-sources</phase>
352                     <goals>
353                        <goal>unpack-dependencies</goal>
354                     </goals>
355                     <configuration>
356                        <includeArtifactIds>
357                           tck-TestModule1,
358                           tck-TestModule2,
359                           tck-TestModule3,
360                           tck-V2AnnotationTests,
361 ...
362                           tck-V3PortletContextTests
363
364                        </includeArtifactIds>
365                        <includes>${test.file.dir}/*.xml</includes>
366                        <outputDirectory>${project.build.directory}</outputDirectory>
367                     </configuration>
368                  </execution>
369               </executions>
370            </plugin>
```

Also in the driver POM, add the **artifact ID** to the includeArtifactIds list in the maven dependency plugin configuration.

Adding a New Module to the TCK Build



In the driver subproject, edit the pageFiles.xml and testfiles.xml files.

```
64    <fl:file>tck-V2SigTestsWrapper-pages.xml</fl:file>
65
66    <fl:file>tck-V3PortletContextTests-pages.xml</fl:file>
67
68 </fl:filelist>
```

Add the name of the file containing the page(s) for your module to the filelist element in pageFiles.xml. The name is constructed from the artifact ID for your module concatenated with the suffix '-pages.xml'

```
64    <fl:file>tck-V2SigTestsWrapper-tests.xml</fl:file>
65
66    <fl:file>tck-V3PortletContextTests-tests.xml</fl:file>
67
68 </fl:filelist>
```

Add the name of the file containing the test cases for your module to the filelist element in testFiles.xml. The name is constructed from the artifact ID for your module concatenated with the suffix '-tests.xml'